



CS 225

Data Structures

October 26 – Graph Traversals

G Carl Evans



mp_mazes AMA

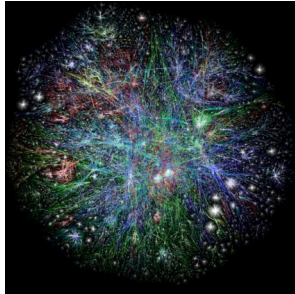
Today at 7pm on Zoom

Meeting ID: 820 2669 7494

Passcode: 139341

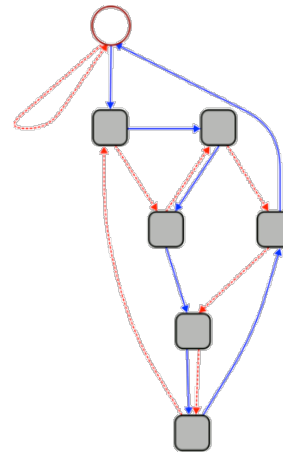
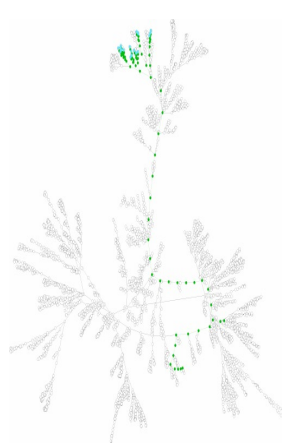
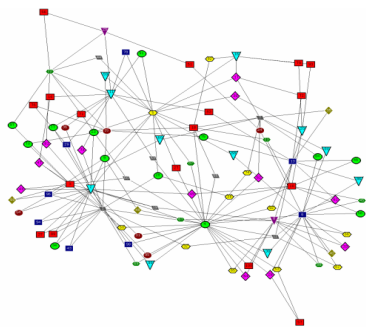
<https://illinois.zoom.us/j/82026697494?pwd=OVM0SWdPTCtKSjZ1TXBKUE10MmJYQT09>

Graphs



HAMLET

TROILUS AND CRESSIDA



```
heapify(int*, unsigned int):  
  push rbp  
  mov rsi, rbp  
  sub rbp, 30  
  mov dword ptr [rbp + 8], rsi  
  mov dword ptr [rbp + 12], esi  
  jmp .LBB_4
```

```
heapify(int*, unsigned int):  
  mov rax, qword ptr [rbp - 8]  
  mov ecx, dword ptr [rbp - 12]  
  mov edx, ecx  
  mov rax, qword ptr [rax + 4*edx]  
  mov rax, qword ptr [rbp - 8]  
  mov esi, dword ptr [rbp - 12]  
  shr esi, 1  
  mov esi, esi  
  mov ecx, esi  
  mov ecx, dword ptr [rax + 4*edx]  
  jmp .LBB_3
```

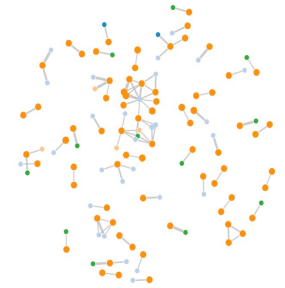
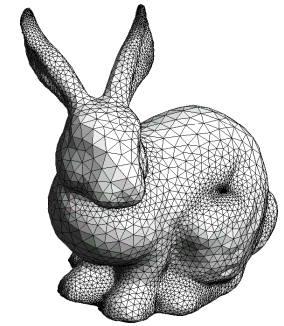
```
heapify(int*, unsigned int):  
  mov rax, qword ptr [rbp - 8]  
  mov rax, dword ptr [rbp - 8]  
  mov ecx, dword ptr [rbp - 12]  
  mov ecx, ecx  
  mov rax, qword ptr [rax + 4*ecx]  
  mov rax, qword ptr [rbp - 8]  
  mov esi, dword ptr [rbp - 12]  
  shr esi, 1  
  mov esi, esi  
  mov ecx, esi  
  mov ecx, dword ptr [rax + 4*ecx]  
  mov rsi, qword ptr [rbp - 8]  
  mov ecx, dword ptr [rbp - 12]  
  shr ecx, 1  
  mov esi, ecx  
  call heapify(int*, unsigned int)
```

```
.LBB_3:  
  jmp .LBB_4
```

```
.LBB_4:  
  add rbp, 30  
  pop rbp  
  ret
```

To study all of these structures:

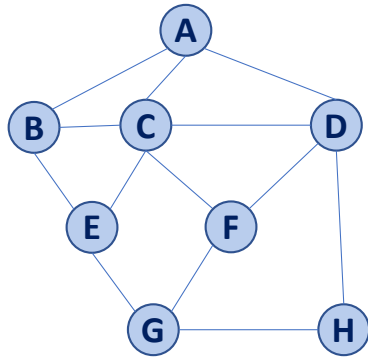
1. A common vocabulary
2. Graph implementations
3. Graph traversals
4. Graph algorithms



```
1 BFS(G) :
2   Input: Graph, G
3   Output: A labeling of the edges on
4           G as discovery and cross edges
5
6   foreach (Vertex v : G.vertices()):
7       setLabel(v, UNEXPLORED)
8   foreach (Edge e : G.edges()):
9       setLabel(e, UNEXPLORED)
10  foreach (Vertex v : G.vertices()):
11      if getLabel(v) == UNEXPLORED:
12          BFS(G, v)
```

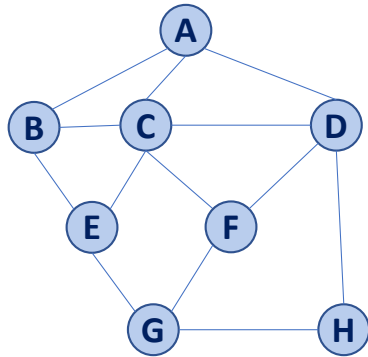
```
14 BFS(G, v) :
15   Queue q
16   setLabel(v, VISITED)
17   q.enqueue(v)
18
19   while !q.empty():
20       v = q.dequeue()
21       foreach (Vertex w : G.adjacent(v)):
22           if getLabel(w) == UNEXPLORED:
23               setLabel(v, w, DISCOVERY)
24               setLabel(w, VISITED)
25               q.enqueue(w)
26           elseif getLabel(v, w) == UNEXPLORED:
27               setLabel(v, w, CROSS)
```

Traversal: BFS



v	d	P	Adjacent Edges
A			
B			
C			
D			
E			
F			
G			
H			

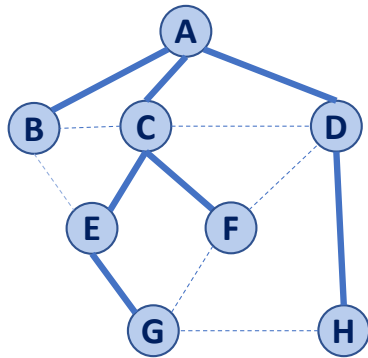
Traversal: BFS



v	d	P	Adjacent Edges
A	0	-	C B D
B			A C E
C			B A D E F
D			A C F H
E			B C G
F			C D G
G			E F H
H			D G

A

Traversal: BFS



v	d	P	Adjacent Edges
A	0	-	C B D
B	1	A	A C E
C	1	A	B A D E F
D	1	A	A C F H
E	2	C	B C G
F	2	C	C D G
G	3	E	E F H
H	2	D	D G

~~G H F E D B C A~~



BFS Analysis

Q: Does our implementation handle disjoint graphs?
If so, what code handles this?

- *How do we use this to count components?*

Q: Does our implementation detect a cycle?

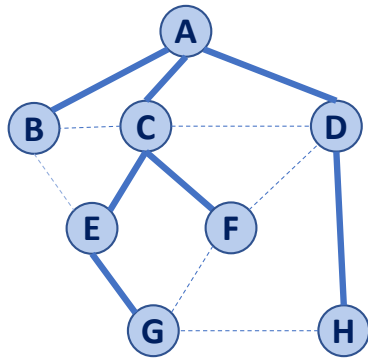
- *How do we update our code to detect a cycle?*

Q: What is the running time?


```
1 BFS(G) :
2   Input: Graph, G
3   Output: A labeling of the edges on
4           G as discovery and cross edges
5
6   foreach (Vertex v : G.vertices()):
7       setLabel(v, UNEXPLORED)
8   foreach (Edge e : G.edges()):
9       setLabel(e, UNEXPLORED)
10  foreach (Vertex v : G.vertices()):
11      if getLabel(v) == UNEXPLORED:
12          BFS(G, v)
```

```
14 BFS(G, v) :
15   Queue q
16   setLabel(v, VISITED)
17   q.enqueue(v)
18
19   while !q.empty():
20       v = q.dequeue()
21       foreach (Vertex w : G.adjacent(v)):
22           if getLabel(w) == UNEXPLORED:
23               setLabel(v, w, DISCOVERY)
24               setLabel(w, VISITED)
25               q.enqueue(w)
26           elseif getLabel(v, w) == UNEXPLORED:
27               setLabel(v, w, CROSS)
```

Running time of BFS



While-loop at **:19**?

For-loop at **:21**?

v	d	P	Adjacent Edges
A	0	-	C B D
B	1	A	A C E
C	1	A	B A D E F
D	1	A	A C F H
E	2	C	B C G
F	2	C	C D G
G	3	E	E F H
H	2	D	D G



BFS Observations

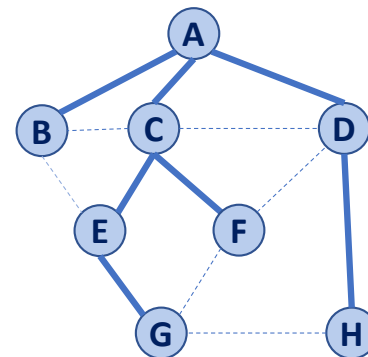
Q: What is a shortest path from **A** to **H**?

Q: What is a shortest path from **E** to **H**?

Q: How does a cross edge relate to **d**?

Q: What structure is made from discovery edges?

v	d	P	Adjacent Edges
A	0	-	C B D
B	1	A	A C E
C	1	A	B A D E F
D	1	A	A C F H
E	2	C	B C G
F	2	C	C D G
G	3	E	E F H
H	2	D	D G





BFS Observations

Obs. 1: BFS can be used to count components.

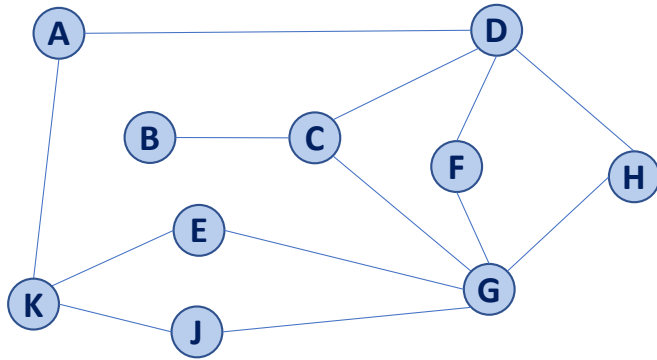
Obs. 2: BFS can be used to detect cycles.

Obs. 3: In BFS, **d** provides the shortest distance to every vertex.

Obs. 4: In BFS, the endpoints of a cross edge never differ in distance, **d**, by more than 1:

$$|d(u) - d(v)| = 1$$

Traversal: DFS



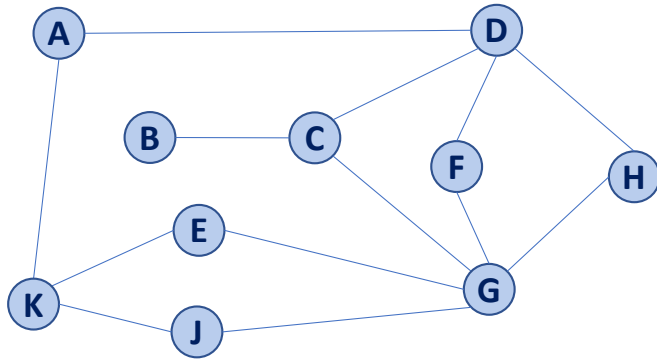
```
1 BFS(G) :
2   Input: Graph, G
3   Output: A labeling of the edges on
4           G as discovery and cross edges
5
6   foreach (Vertex v : G.vertices()):
7       setLabel(v, UNEXPLORED)
8   foreach (Edge e : G.edges()):
9       setLabel(e, UNEXPLORED)
10  foreach (Vertex v : G.vertices()):
11      if getLabel(v) == UNEXPLORED:
12          BFS(G, v)
```

```
14 BFS(G, v) :
15   Queue q
16   setLabel(v, VISITED)
17   q.enqueue(v)
18
19   while !q.empty():
20       v = q.dequeue()
21       foreach (Vertex w : G.adjacent(v)):
22           if getLabel(w) == UNEXPLORED:
23               setLabel(v, w, DISCOVERY)
24               setLabel(w, VISITED)
25               q.enqueue(w)
26           elseif getLabel(v, w) == UNEXPLORED:
27               setLabel(v, w, CROSS)
```

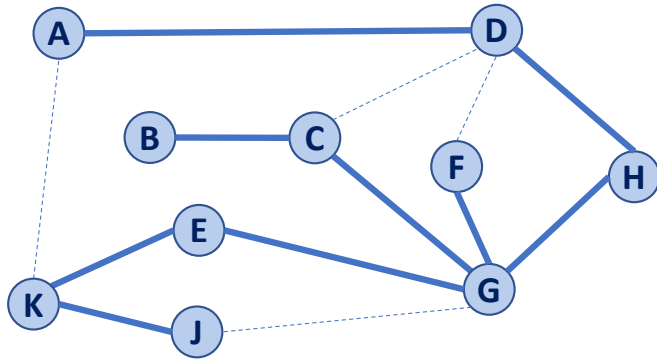
```
1 DFS(G) :
2   Input: Graph, G
3   Output: A labeling of the edges on
4           G as discovery and back edges
5
6   foreach (Vertex v : G.vertices()):
7     setLabel(v, UNEXPLORED)
8   foreach (Edge e : G.edges()):
9     setLabel(e, UNEXPLORED)
10  foreach (Vertex v : G.vertices()):
11    if getLabel(v) == UNEXPLORED:
12      DFS(G, v)
```

```
14 DFS(G, v) :
15 Queue q
16   setLabel(v, VISITED)
17 q.enqueue(v)
18
19 while !q.empty():
20 v = q.dequeue()
21   foreach (Vertex w : G.adjacent(v)):
22     if getLabel(w) == UNEXPLORED:
23       setLabel(v, w, DISCOVERY)
24       setLabel(w, VISITED)
25       DFS(G, w)
26     elseif getLabel(v, w) == UNEXPLORED:
27       setLabel(v, w, BACK)
```

Traversal: DFS



Traversal: DFS



————— Discovery Edge

----- Back Edge

```
1 BFS(G) :
2   Input: Graph, G
3   Output: A labeling of the edges on
4           G as discovery and cross edges
5
6   foreach (Vertex v : G.vertices()):
7       setLabel(v, UNEXPLORED)
8   foreach (Edge e : G.edges()):
9       setLabel(e, UNEXPLORED)
10  foreach (Vertex v : G.vertices()):
11      if getLabel(v) == UNEXPLORED:
12          BFS(G, v)
```

```
14 BFS(G, v) :
15   Queue q
16   setLabel(v, VISITED)
17   q.enqueue(v)
18
19   while !q.empty():
20       v = q.dequeue()
21       foreach (Vertex w : G.adjacent(v)):
22           if getLabel(w) == UNEXPLORED:
23               setLabel(v, w, DISCOVERY)
24               setLabel(w, VISITED)
25               q.enqueue(w)
26           elseif getLabel(v, w) == UNEXPLORED:
27               setLabel(v, w, CROSS)
```

```
1 DFS(G) :
2   Input: Graph, G
3   Output: A labeling of the edges on
4           G as discovery and back edges
5
6   foreach (Vertex v : G.vertices()):
7     setLabel(v, UNEXPLORED)
8   foreach (Edge e : G.edges()):
9     setLabel(e, UNEXPLORED)
10  foreach (Vertex v : G.vertices()):
11    if getLabel(v) == UNEXPLORED:
12      DFS(G, v)
```

```
14 DFS(G, v) :
15 Queue q
16   setLabel(v, VISITED)
17 q.enqueue(v)
18
19 while !q.empty():
20 v = q.dequeue()
21   foreach (Vertex w : G.adjacent(v)) :
22     if getLabel(w) == UNEXPLORED:
23       setLabel(v, w, DISCOVERY)
24       setLabel(w, VISITED)
25       DFS(G, w)
26     elseif getLabel(v, w) == UNEXPLORED:
27       setLabel(v, w, BACK)
```

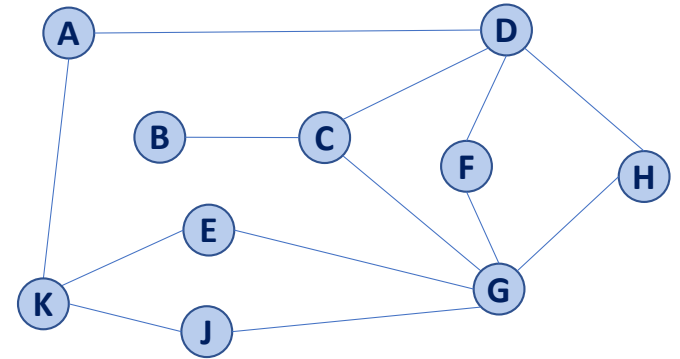
Running time of DFS

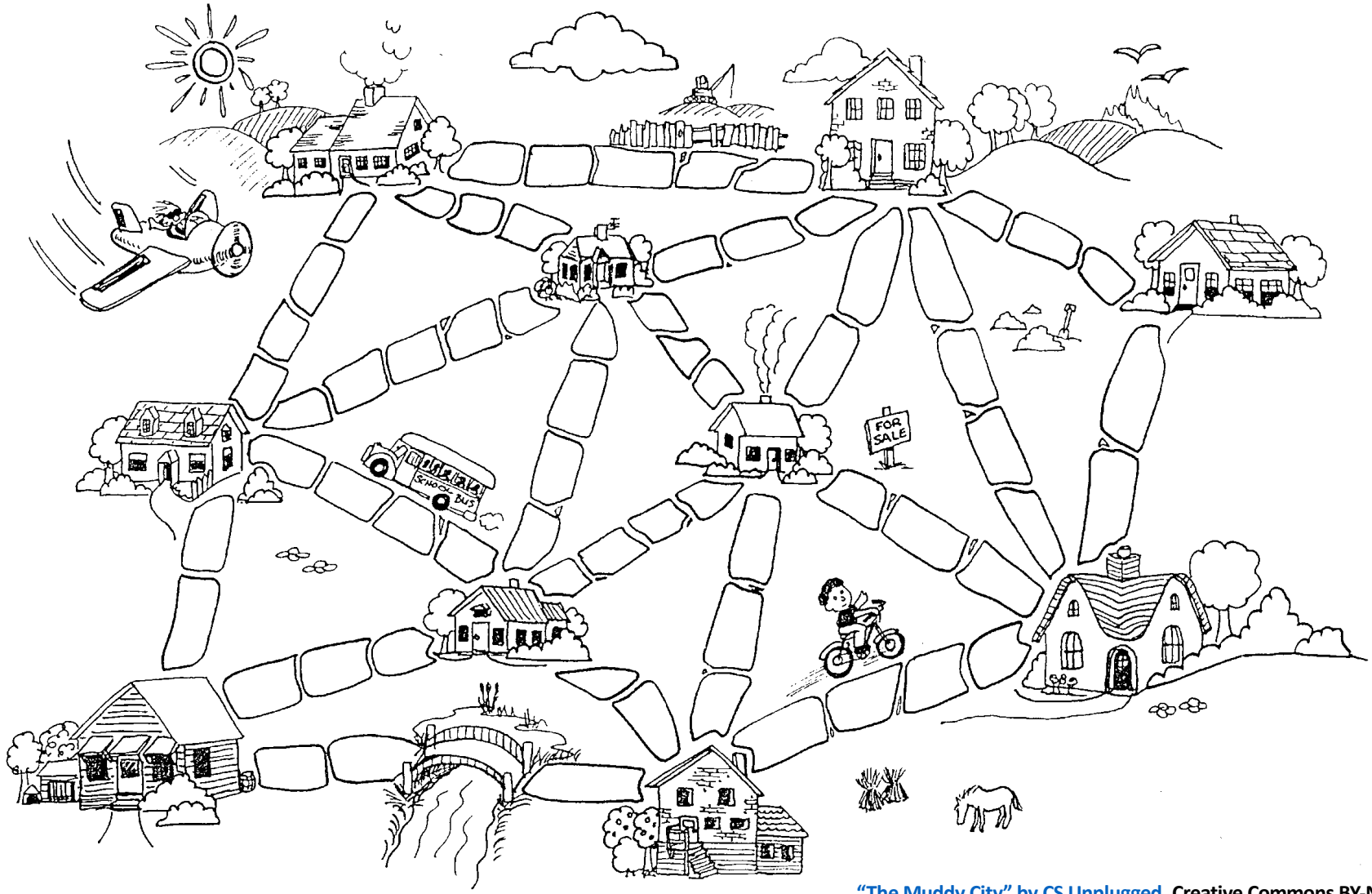
Labeling:

- Vertex:
- Edge:

Queries:

- Vertex:
- Edge:





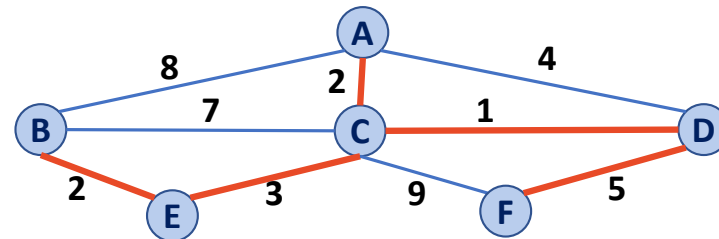
["The Muddy City"](#) by CS Unplugged, Creative Commons BY-NC-SA 4.0

Minimum Spanning Tree Algorithms

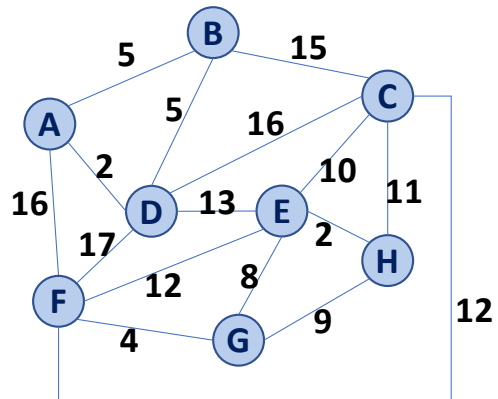
Input: Connected, undirected graph G with edge weights (unconstrained, but must be additive)

Output: A graph G' with the following properties:

- G' is a spanning graph of G
- G' is a tree (connected, acyclic)
- G' has a minimal total weight among all spanning trees

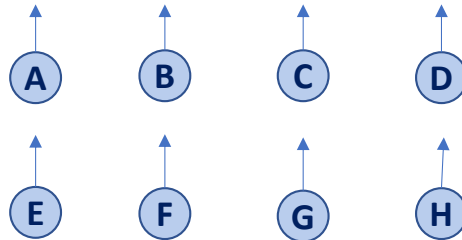
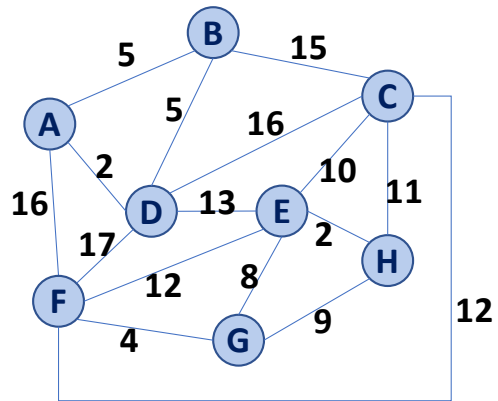


Kruskal's Algorithm



(A, D)
(E, H)
(F, G)
(A, B)
(B, D)
(G, E)
(G, H)
(E, C)
(C, H)
(E, F)
(F, C)
(D, E)
(B, C)
(C, D)
(A, F)
(D, F)

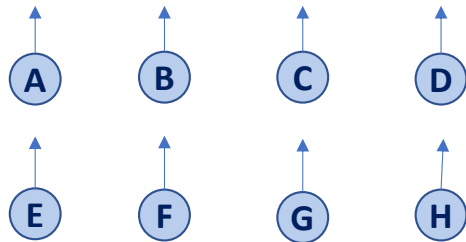
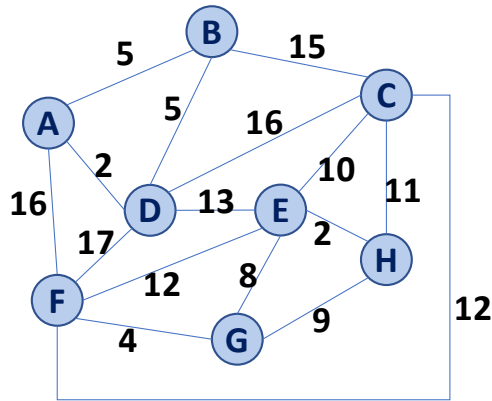
Kruskal's Algorithm



(A, D)
(E, H)
(F, G)
(A, B)
(B, D)
(G, E)
(G, H)
(E, C)
(C, H)
(E, F)
(F, C)
(D, E)
(B, C)
(C, D)
(A, F)
(D, F)

Kruskal's Algorithm

(A, D)
(E, H)
(F, G)
(A, B)
(B, D)
(G, E)
(G, H)
(E, C)
(C, H)
(E, F)
(F, C)
(D, E)
(B, C)
(C, D)
(A, F)
(D, F)



```

1 KruskalMST(G):
2   DisjointSets forest
3   foreach (Vertex v : G):
4     forest.makeSet(v)
5
6   PriorityQueue Q // min edge weight
7   foreach (Edge e : G):
8     Q.insert(e)
9
10  Graph T = (V, {})
11
12  while |T.edges()| < n-1:
13    Vertex (u, v) = Q.removeMin()
14    if forest.find(u) == forest.find(v):
15      T.addEdge(u, v)
16      forest.union( forest.find(u),
17                  forest.find(v) )
18
19  return T

```

Kruskal's Algorithm

Priority Queue:	Heap	Sorted Array
Building :7-9		
Each removeMin :13		

```
1 KruskalMST(G):
2   DisjointSets forest
3   foreach (Vertex v : G):
4     forest.makeSet(v)
5
6   PriorityQueue Q // min edge weight
7   foreach (Edge e : G):
8     Q.insert(e)
9
10  Graph T = (V, {})
11
12  while |T.edges()| < n-1:
13    Vertex (u, v) = Q.removeMin()
14    if forest.find(u) == forest.find(v):
15      T.addEdge(u, v)
16      forest.union( forest.find(u),
17                  forest.find(v) )
18
19  return T
```

Kruskal's Algorithm

Priority Queue:	Total Running Time
Heap	
Sorted Array	

```
1 KruskalMST(G) :
2   DisjointSets forest
3   foreach (Vertex v : G) :
4     forest.makeSet(v)
5
6   PriorityQueue Q    // min edge weight
7   foreach (Edge e : G) :
8     Q.insert(e)
9
10  Graph T = (V, {})
11
12  while |T.edges()| < n-1:
13    Vertex (u, v) = Q.removeMin()
14    if forest.find(u) == forest.find(v) :
15      T.addEdge(u, v)
16      forest.union( forest.find(u),
17                  forest.find(v) )
18
19  return T
```